**FSD Lab Assignment 1**

**Name: - Hemanshu Vaidya Batch: -E1**

**Roll No: - 29 Panel: - E**

**Problem Statement:**

Created a public git repository for your team and submit the repo URL as a solution to this assignment, Learn Git concept of Local and Remote Repository, Push, Pull, Merge and Branch.

**Aim:**

Version control with Git.

**Objectives:**

1. To introduce the concepts and software behind version control, using the example of Git.
2. To understand the use of 'version control' in the context of a coding project.
3. To learn Git version control with Clone, commit to, and push, pull from a git repository.

**#Theory:**

1. **What is Git? What is Version Control?**

-> Git is a distributed version control system (DVCS) used for tracking changes in computer files and coordinating work on those files among multiple people or teams. It was created by Linus Torvalds in 2005 and has since become the most widely used version control system in the world.

Version control, in general, is a system that records changes to a file or set of files over time, so you can recall specific versions later. It allows you to:

Track Changes: Version control systems keep a detailed history of changes made to a file, including who made the change and when it was made.

Collaborate: Multiple people can work on the same project simultaneously without interfering with each other's work. They can merge their changes together seamlessly.

Revert to Previous States: If a mistake is made or if you want to go back to a previous version of a file or project, version control makes it easy to revert.

Branching and Merging: Version control systems, like Git, enable the creation of branches where you can work on new features or changes separately from the main codebase. You can later merge these changes back into the main codebase.

Documentation: Version control systems often encourage good documentation practices because each change is associated with a commit message explaining the purpose of the change.

Git, specifically, is known for its speed, flexibility, and strong branching and merging capabilities. It's widely used in software development to manage source code, but it's also used in many other fields for tracking changes to documents, configurations, and other types of files.

1. **How to use Git for version controlling?**

-> Install Git:

Start by downloading and installing Git on your computer. This is a one-time setup process. You can find installation instructions for your operating system on the official Git website.

Configure Git:

After installing Git, you should configure it with your name and email address. This information helps identify the author of commits, so it's important to set it up correctly.

Create a Git Repository:

To begin version controlling a project, navigate to the directory where your project resides. Use the git init command to initialize a new Git repository in that directory.

Add Files to the Repository:

Start tracking files by adding them to the Git repository. You can add individual files or entire directories using the git add command. This step prepares the files for a commit.

Commit Changes:

Committing is like taking a snapshot of your project at a particular point in time. Each commit has a commit message that explains the changes made in that snapshot. Use the git commit command to create a commit with a message.

Viewing Status and History:

You can check the status of your working directory using git status. This shows you which files have changes and which are ready to be committed. To view the commit history of your project, use git log.

Create Branches:

Branches are used to work on different features or tasks independently. Create a new branch with git branch branch-name. This allows you to make changes without affecting the main project until you're ready to merge.

Switch Branches:

To switch to a different branch, use git checkout branch-name. This lets you work on that branch and make changes specific to it.

Merge Branches:

Once you've completed work on a branch and want to integrate those changes into the main project, you can merge the branch back into the main branch. Use git merge branch-name while on the main branch to accomplish this.

Push and Pull:

When working with remote repositories, like on platforms such as GitHub or GitLab, you'll use git push to send your local changes to the remote repository and git pull to fetch changes made by others and update your local repository.

These are the essential steps and concepts for using Git for version control. Remember that Git is a powerful tool with many advanced features and workflows, but these basics should help you get started with tracking changes and collaborating on projects effectively. Practice and familiarity with Git will make you more proficient over time.

**#FAQs:**

1. **What is branching in Git?**

->Branching in Git is a fundamental concept that allows you to diverge from the main line of development (often referred to as the "master" or "main" branch) and work on a separate line of development without affecting the main codebase. Each branch represents an independent line of development with its own set of changes. Branches are useful for:

Isolation: Branches allow you to work on new features, bug fixes, or experiments without interfering with the stability of the main project. This isolation helps prevent conflicts and errors in the main codebase.

Parallel Development: Multiple team members can work on different branches simultaneously. This parallel development speeds up the development process and enables teams to collaborate effectively.

Testing and Experimentation: You can create branches to test new ideas, features, or fixes without risking the integrity of the main project. If an experiment doesn't work out, you can simply discard the branch.

Here are some key branching concepts in Git:

Main Branch: This is typically the default and primary branch (e.g., "master" or "main"). It represents the stable version of the project.

Feature Branch: A feature branch is created to work on a specific feature or enhancement. Developers work on this branch until the feature is complete and ready for integration into the main branch.

Bugfix Branch: Similar to a feature branch, a bugfix branch is created to fix a specific issue or bug in the code. Once the bug is fixed, the branch can be merged back into the main branch.

Release Branch: Before releasing a new version of your software, you can create a release branch. This branch is used for final testing and last-minute bug fixes. Once everything is stable, the release branch can be merged into the main branch and tagged with a version number.

Hotfix Branch: In cases where critical bugs or security issues are discovered in the main branch, a hotfix branch can be created to address these issues urgently. The changes are then merged into both the main branch and any active feature branches.

Here are some common Git commands for branching:

git branch: Lists all branches in the repository, with the current branch highlighted.

git branch branch-name: Creates a new branch with the specified name.

git checkout branch-name: Switches to the specified branch.

git merge branch-name: Merges changes from the specified branch into the current branch.

git branch -d branch-name: Deletes the specified branch (use -d for a safe delete that requires the branch to be fully merged).

git branch -D branch-name: Forces the deletion of the specified branch, even if it's not fully merged (use with caution).

Branching is a powerful and essential feature of Git that facilitates collaborative and organized software development. It allows you to manage complex projects and workflows effectively while maintaining version control and code stability.

1. **How to create and merge branches in Git? Write the commands used.**

-> Create a New Branch:

Create a new branch using the git branch branch-name command. For example, to create a branch called "feature-branch," you'd run git branch feature-branch.

Switch to the New Branch:

Switch to the newly created branch using the git checkout branch-name command. You can combine branch creation and checkout in one step with git checkout -b branch-name.

Make Changes:

Make your code changes, additions, or modifications in this branch.

Add and Commit:

Add your changes to the staging area with git add ., followed by committing the changes with git commit -m "Your commit message here".

Switch Back to the Main Branch:

Before merging your feature branch, switch back to the main branch using git checkout main (or whatever your main branch is named).

Merge the Feature Branch:

Merge the feature branch into the main branch using git merge feature-branch. This command incorporates the changes from the feature branch into the main branch.

Resolve Conflicts (if any):

If there are conflicts between the changes in the feature branch and the main branch, you'll need to resolve them manually. Git will guide you through the process.

Push Changes (if using a remote repository):

If you're working with a remote Git repository (e.g., GitHub, GitLab), push the changes to the remote repository using git push origin main.

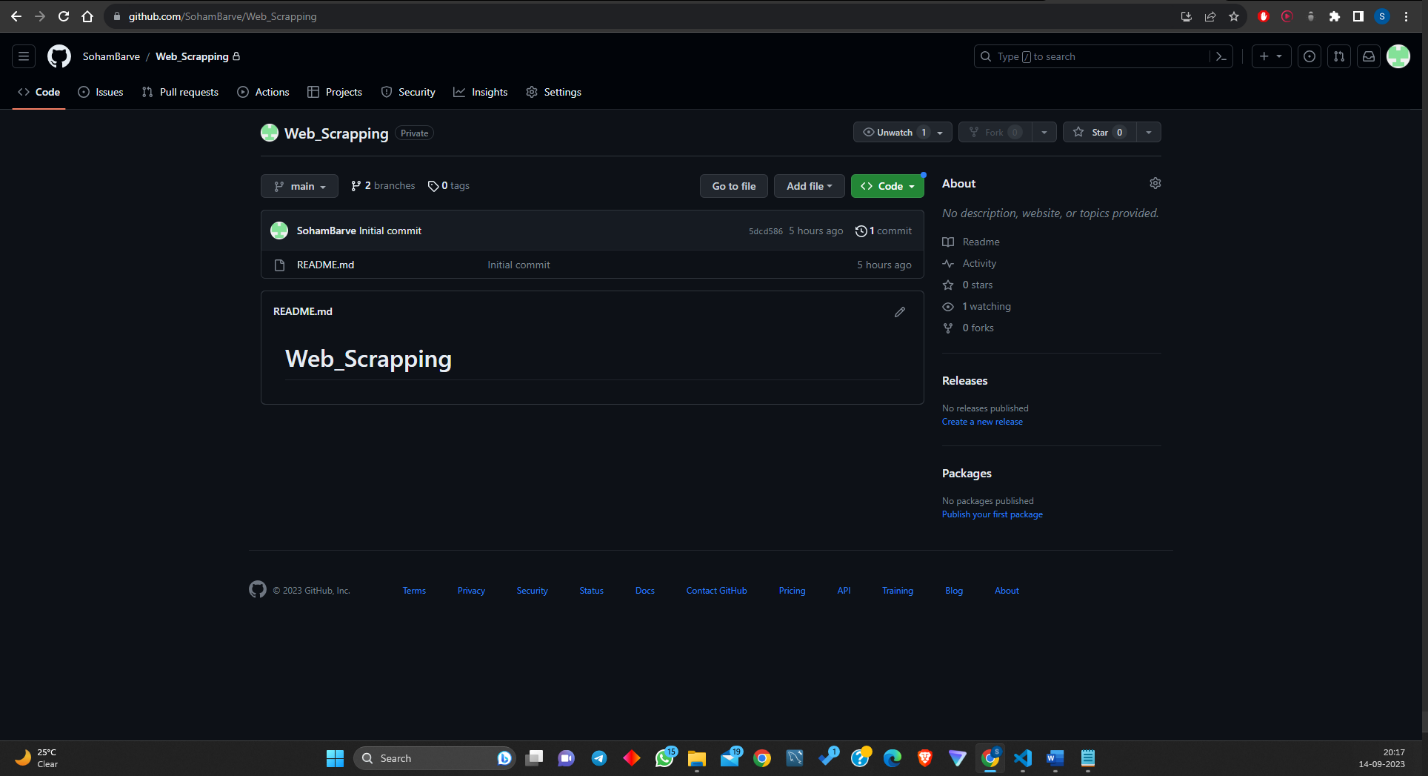
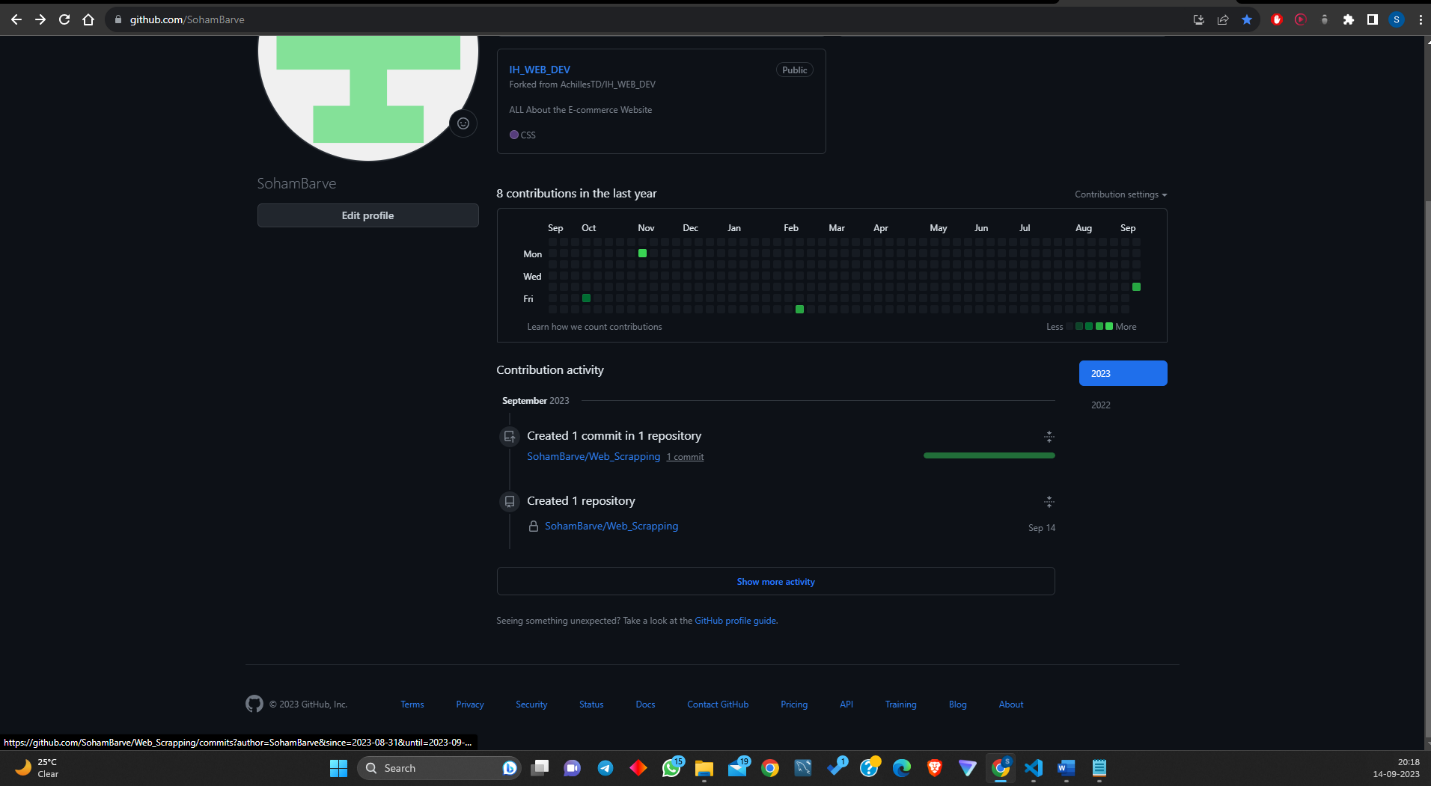
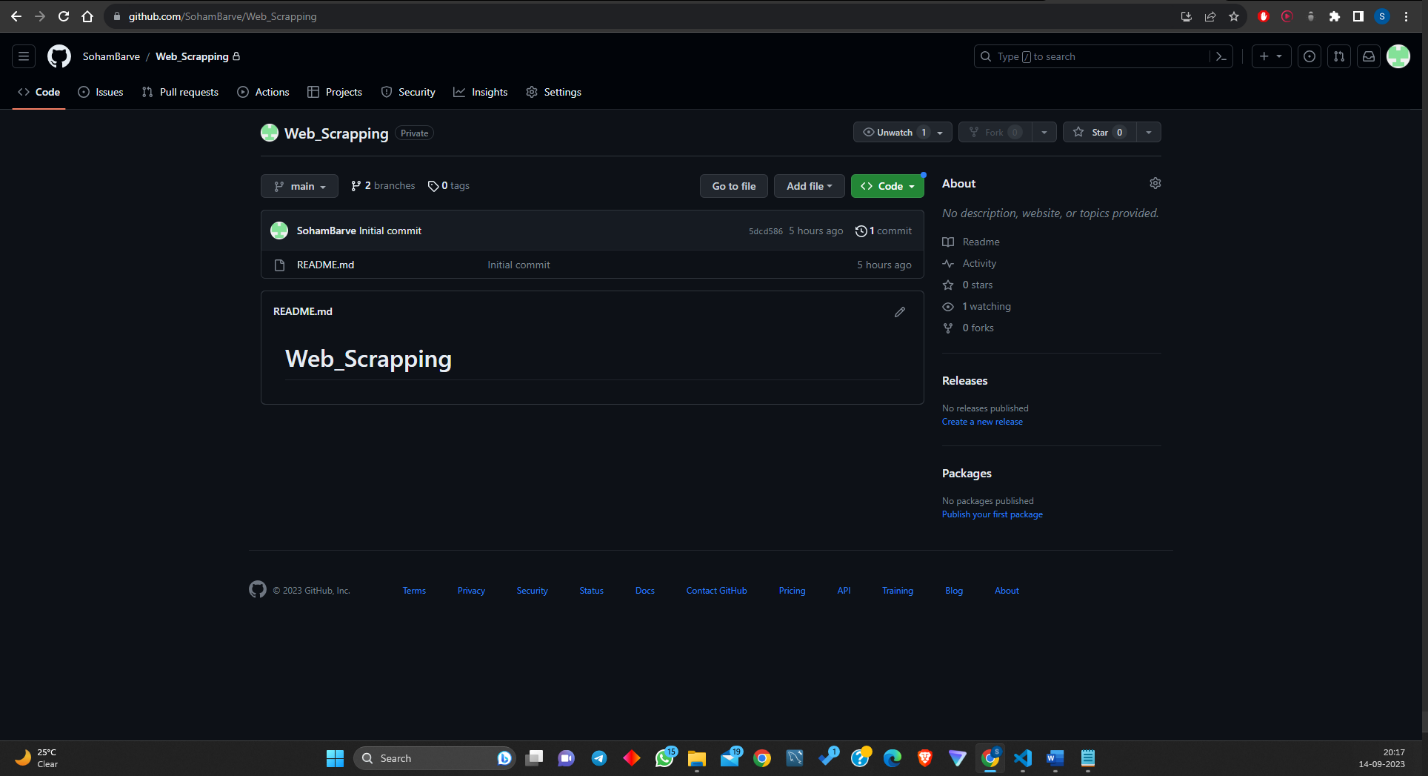
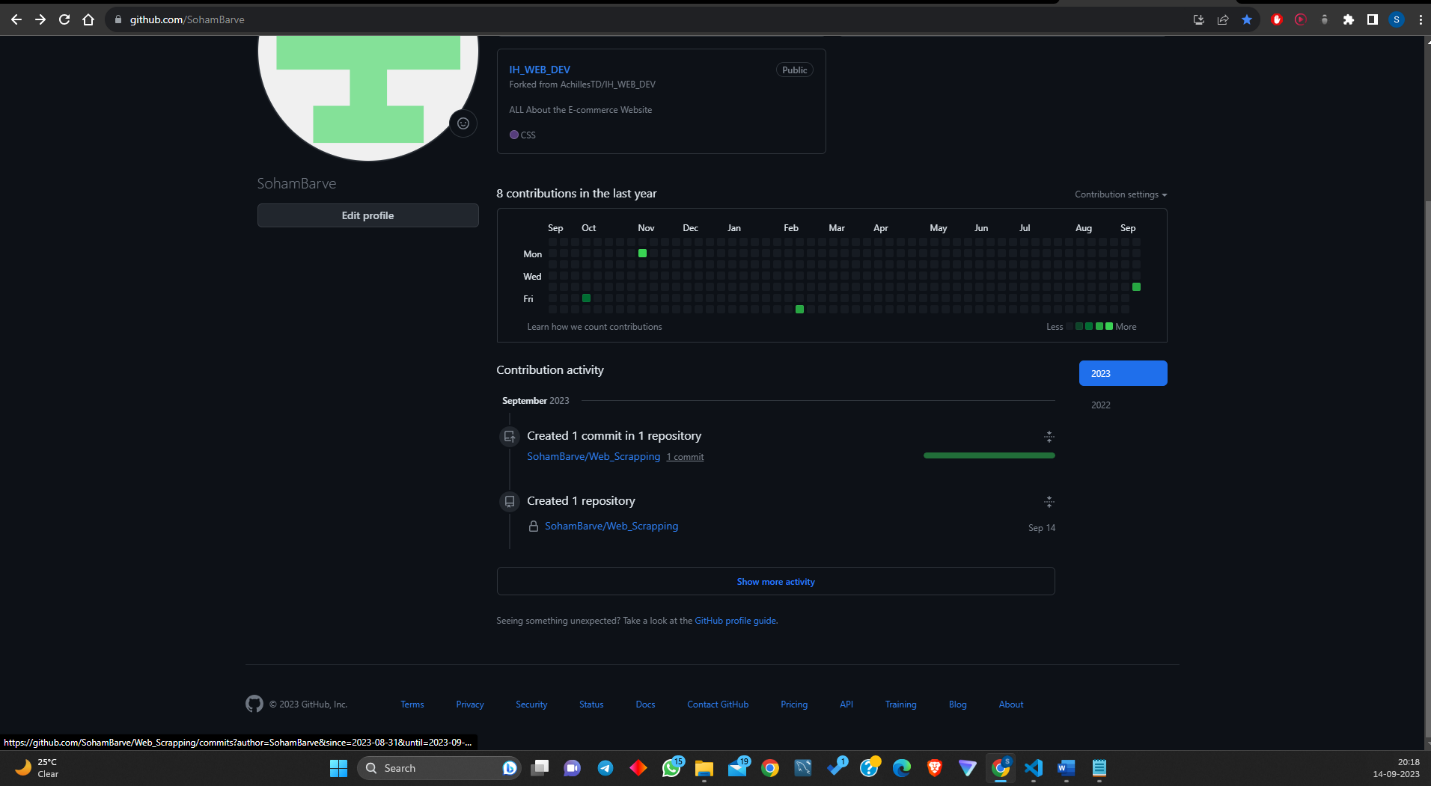
Delete the Feature Branch (optional):

If you no longer need the feature branch after merging, you can delete it with git branch -d feature-branch for a safe delete that checks for full merging. If you're sure you want to delete it, use git branch -D feature-branch.

This process allows you to create, work on, and merge feature branches in Git, keeping your main branch clean and organized while developing new features or making changes.

**Output:**

Screenshots

**URL :-**